Constraint Programming for Association Rules

Mohamed-Bachir Belaid* Christian Bessiere* Nadjib Lazaar*

Abstract

Discovering association rules among items in a dataset is one of the fundamental problems in data mining. It has recently been shown that constraint programming is a flexible way to tackle data mining tasks. In this paper we propose a declarative model based on constraint programming to capture association rules. Our model also allows us to specify any additional property and/or user's constraints on the kind of rules the user is looking for. To implement our model, we introduce a new global constraint, CONFIDENT, for ensuring the confidence of rules. We prove that completely propagating CONFIDENT is NP-hard. We thus provide a decomposition of CONFIDENT. In addition to user's constraints on the items composing body and head of the rules, we show that we can capture the popular minimal non-redundant property of association rules. An experimental analysis shows the practical effectiveness of our approach compared to existing approaches.

1 Introduction

Mining association rules aims at discovering interesting regularities between items in large-scale datasets. Association rules (ARs) were originally introduced by Agrawal et al. [1] for sales transactions and products. An association rule captures an information of the kind *"if we have A and B, the chances to have C are high"*. Nowadays, a broad spectrum of application domains ask for this kind of information with a variety of datasets.

In practice, the number of rules is often huge and can easily exceed the size of the dataset itself. In this case the user faces an enormous number of irrelevant rules. However, most of the time the user is interested in rules that satisfy some specific properties. For instance, the user may ask for rules containing/not containing some specific items [11], or rules with a body and/or head of a given size, etc. The user may also ask for some *representative rules* according to a condensed representation. The cover operator allows us to derive a set of rules from a representative one [8]. A stronger version of representative rules consists in looking for rules with minimum body maximum head [9]. Finally, the most common notion of representative rules are the *minimal non-redundant association rules* (MNR) [2]. A rule is an MNR if there does not exist any other rule with the same support and the same confidence that is obtained by removing items from the body or adding items to the head.

Several specialized algorithms have been proposed to discover ARs and MNRs. For ARs, the process starts first by discovering all frequent itemsets and then generating all ARs. Discovering MNRs can be reduced to seeking the set of frequent closed itemsets (FCIs) and their frequent generator itemsets (FGIs). The specialized algorithms differ in the way to reach a matching between FCIs and FGIs. ZART [14] is a refined version of PASCAL [3] with an optimal way to link an FGI to its FCI. The hybrid algorithm ECLAT-Z [15] combines ZART with the vertical representation of the dataset of the algorithm ECLAT. ECLAT-Z acts in three steps. First, it uses the vertical algorithm ECLAT for extracting all frequent itemsets. Second, it filters out the set of frequent itemsets to get FCIs and FGIs. Then, it associates FGIs to their FCIs. Such hybridization makes ECLAT-Z one of the fastest AR mining algorithms.

Nevertheless, looking for ARs with additional user-specified constraints remains a bottleneck. According to Wojciechowski and Zakrzewicz [16], there are three ways to handle the additional user's constraints. We can use a pre-processing step that restricts the dataset to only transactions that satisfy the constraints. Such a technique cannot be used on all kinds of constraints. We can integrate the filtering of the user's constraints into the specialized data mining process in order to extract only the ARs satisfying the constraints. Such a technique requires the development of a new algorithm for each new AR problem with user's constraints. We can finally use a post-processing step to filter out the ARs violating the user's constraints. Such a brute-force technique can be computationally infeasible when the problem without the user's constraints has too many solutions.

In a recent line of work [7, 10, 13], constraint programming (CP) has been used as a declarative way to solve some data mining tasks, such as itemset mining or sequence mining. Such an approach has

*LIRMM, University of Montpellier, CNRS, Montpellier, France, {belaid, bessiere, lazaar}@lirmm.fr not competed yet with state of the art data mining algorithms in terms of CPU time for standard data mining queries but the CP approach is competitive as soon as we need to add user's constraints. In addition, adding constraints is easily done by specifying the constraints directly in the model without the need to revise the solving process. However, the CP approach has not yet been applied to association rules.

The mining of association rules can be formulated as a propositional satisfiability problem (SAT) with some linear inequalities to ensure the minimum frequency and the minimum confidence of a given rule [5, 6]. The solving process is based on a pseudo-Boolean solver.

In this paper, we propose a full CP model for finding ARs. We use global constraints to ensure frequency and confidence of the rules. We take the frequency global constraint from existing literature (i.e., CoverSize constraint [13]) but we need to introduce a new global constraint Confident for ensuring the minimum confidence of the extracted rules. We show that domain consistency on Confident is NP-hard. We then propose a decomposition of Confident. We show that our CP model can easily be extended for taking into account any kind of user's constraints, such as cardinality of the rule, mandatory or forbidden items in the rule, etc. We show that our CP model is also able to capture the notion of MNR. Experiments on several known large-scale datasets show the effectiveness of our CP model.

The paper is organized as follows. Section 2 gives some background material. Section 3 presents our CP model for computing association rules. Section 4 defines the global constraint Confident, a global constraint for ensuring the confidence of a rule. Section 5 presents the extension of our CP model to compute MNRs. The global constraint Generator, which is needed in that model, is defined and a propagator is proposed. Section 6 reports experiments. Finally, we conclude in Section 7.

2 Background

2.1 Itemsets Let $\mathcal{I} = \{1, \dots, n\}$ be a set of n *item* indices and $\mathcal{T} = \{1, \dots, m\}$ a set of m *transaction* indices. An itemset P is a non-empty subset of $\mathcal{I}$. $\mathcal{D} = \{t_1, \dots, t_m\}$ is the transactional dataset, where for all $i \in \mathcal{T}$, t_i is an itemset. The cover of an itemset P, denoted by $cover(P)$, is the set of transactions containing P. The (relative) frequency of an itemset P is $freq(P) = \frac{|cover(P)|}{|\mathcal{T}|}$. Let $s \in [0..1]$ be some given constant called a *minimum support*. An itemset P is frequent if $freq(P) \geq s$. An itemset P is *closed* if and only if there does not exist any itemset $Q \supsetneq P$ such that $freq(Q) = freq(P)$. A *generator* is an itemset P such that there does not exist any itemset $Q \subsetneq P$ such that $freq(Q) = freq(P)$. Generators were introduced for efficiently mining frequent itemsets [3].

Table 1: Transaction dataset example with six items and five transactions.

trans.	Items					
t_1	A	B				
t_2	A		C	D	E	
t_3		B	C	D		F
t_4	A	B	C	D		
t_5	A	B	C			F

2.2 Association Rules An *association rule* is an implication of the form $X \to Y$, where X and Y are itemsets such that $X \cap Y = \emptyset$ and $Y \neq \emptyset$. X represents the *body* of the rule and Y represents its *head*. The frequency of a rule $X \to Y$ is the frequency of the itemset $X \cup Y$, that is, $freq(X \to Y) = freq(X \cup Y)$. The *confidence* of a rule captures how often Y occurs in transactions containing X, that is, $conf(X \to Y) = \frac{freq(X \to Y)}{freq(X)}$. Given a minimum confidence c, a rule $X \to Y$ is confident if $conf(X \to Y) \geq c$. A rule $X \to Y$ is valid if it is frequent and confident.

Example 1. *Consider the transaction dataset presented in Table 1. The itemset BCD is closed, but CF is not as $freq(CF) = freq(BCF) = 40\%$. The itemset AC is a generator because $freq(AC) = 60\%$ and none of its subsets ($\emptyset$, A, C) have the same frequency ($freq(\emptyset) = 100\%$, $freq(A) = freq(C) = 80\%$). The itemset CD is not a generator because it has the same frequency as one of its subsets: $freq(CD) = freq(D) = 60\%$. If $s = 60\%$ and $c = 70\%$, $C \to A$ is a valid association rule because $freq(C \to A) = 60\%$ and $conf(C \to A) = \frac{freq(C \to A)}{freq(C)} = 75\% \geq 70\%$.*

2.3 Constraint Programming (CP) A *CP model* specifies a set of variables $X = \{x_1, \dots, x_n\}$, a set of domains $dom = \{dom(x_1), \dots, dom(x_n)\}$, where $dom(x_i)$ is the finite set of possible values for x_i, and a set of constraints $\mathcal{C}$ on X. A constraint $c_j \in \mathcal{C}$ is a relation that specifies the allowed combinations of values for its variables $var(c_j)$. An assignment on a set $Y \subseteq X$ of variables is a mapping from variables in Y to values, and a valid assignment is an assignment where all values belong to the domain of their variable. A solution is an assignment on X satisfying all constraints. Constraint programming is the art of writing problems as CP models and solving them by finding solutions. Constraint solvers typically use backtracking search to explore the search space of partial assignments. At each assignment, constraint propagation algorithms (aka, propagators) prune the search space by enforcing local consistency properties such as domain consistency.

A constraint c on $X(c)$ is *domain consistent (DC)* if and only if, for every $x_i \in X(c)$ and every $d_j \in dom(x_i)$, there is a valid assignment satisfying c such that $x_i = d_j$.

Global constraints are constraints defined by a relation on any number of variables. The constraint AllDifferent, specifying that all its variables must take different values is an example of global constraint (see [12]).

EXAMPLE 2. *Consider the following instance of a CP model.* $X = \{x_1, x_2, x_3\}$, $dom(x_1) = \{0, 2\}$, $dom(x_2) = \{0, 2, 4\}$, $dom(x_3) = \{1, 2, 3, 4\}$, *and* $\mathcal{C} = \{x_1 \geq x_2, x_1 + x_2 = x_3\}$. *Value 4 for* x_2 *will be removed by DC because of constraint* $x_1 \geq x_2$. *Values 1 and 3 for* x_3 *will be removed by DC because of constraint* $x_1 + x_2 = x_3$. *This CP model admits the two solutions* $(x_1 = 2, x_2 = 0, x_3 = 2)$ *and* $(x_1 = 2, x_2 = 2, x_3 = 4)$.

3 A CP Model for Association Rules

In this section we present CP-RULE, a CP model for mining association rules. We introduce three vectors x, y and z of n Boolean variables, where x_i, y_i and z_i respectively represent the presence of item i in the body of the rule, in the head of the rule, and in the rule as a whole. In the rest of the paper we will use the following notations:

- $x^{-1}(1) = \{i \in \mathcal{I} \mid dom(x_i) = \{1\}\}$
- $x^{-1}(0) = \{i \in \mathcal{I} \mid dom(x_i) = \{0\}\}$

We use similar notations for vectors y and z. The model CP-RULE should be specified so that for any assignment on x, y, z that is a solution, $x^{-1}(1) \rightarrow y^{-1}(1)$ is a valid association rule.

CP-RULE involves five types of constraints:

$$\text{CP-RULE}_{\mathcal{D},s,c}(x,y,z) = \begin{cases} \forall i \in \mathcal{I}: \ \neg x_i \vee \neg y_i & (1) \\ \bigvee_{i \in \mathcal{I}} \ y_i & (2) \\ \forall i \in \mathcal{I}: \ z_i \iff x_i \vee y_i & (3) \\ \text{CONFIDENT}_{\mathcal{D},c}(x,y) & (4) \\ \text{FREQUENT}_{\mathcal{D},s}(z) & (5) \end{cases}$$

The role of each type of constraint is the following:

(1) ensures that a given item cannot be both in the body and in the head of a rule;

(2) ensures that the head of a rule is not empty;

(3) is a channelling constraint ensuring that $z^{-1}(1) = x^{-1}(1) \cup y^{-1}(1)$;

(4) is a global constraint that ensures that the rule $x^{-1}(1) \rightarrow y^{-1}(1)$ is confident w.r.t. c;

(5) is a global constraint that ensures that $z^{-1}(1)$ is frequent w.r.t s.

The constraint $\text{FREQUENT}_{\mathcal{D},s}$ has already been studied in the literature on itemset mining [7]. The constraint $\text{CONFIDENT}_{\mathcal{D},c}$, however, does not exist yet. We define it in the next section.

4 The Global Constraint Confident

In this section, we present a new global constraint for ensuring the confidence of an association rule. We prove that it is unfortunately NP-hard to enforce domain consistency on this constraint. We finally give a decomposition, using existing constraints, which is semantically equivalent to the CONFIDENT constraint.

DEFINITION 1. (CONFIDENT CONSTRAINT) *Let* x *and* y *be two vectors of Boolean variables. Let* D *be a dataset and* c *a minimum confidence. The constraint* $\text{CONFIDENT}_{\mathcal{D},c}(x,y)$ *holds if and only if* $conf(x^{-1}(1) \rightarrow y^{-1}(1)) \geq c$.

THEOREM 1. *Enforcing domain consistency on the constraint* $\text{CONFIDENT}_{\mathcal{D},c}$ *is NP-hard.*

Proof. From [4] we know that if it is NP-complete to decide whether there exists a valid assignment satisfying a global constraint, then, enforcing domain consistency on this constraint is NP-hard. We then prove that it is NP-complete to decide if there exists a valid assignment for $\text{CONFIDENT}_{D,c}$.
Membership. Given the constraint $\text{CONFIDENT}_{D,c}(x,y)$ and a valid assignment on x and y, we traverse the table $\mathcal{D}$ and compute the size of the covers of the itemsets $x^{-1}(1)$ and $x^{-1}(1) \cup y^{-1}(1)$. This is linear in $|\mathcal{D}|$. We then compute the ratio $\frac{|cover(x^{-1}(1) \cup y^{-1}(1))|}{|cover(x^{-1}(1))|}$ and compare it to c to decide if the assignment satisfies the constraint. This is linear in $log|\mathcal{D}|$.
Completeness. We reduce 3SAT to the problem of deciding if there exists a valid assignment satisfying $\text{CONFIDENT}_{\mathcal{D},c}(x,y)$. Given a 3SAT formula F on the set $V = \{v_1, \ldots, v_n\}$ of Boolean variables, we construct the following instance of the constraint $\text{CONFIDENT}_{\mathcal{D},c}$. For clarity purpose, we denote the items in the transaction table $\mathcal{D}$ by `pos1`, `neg1`, $\cdots$, `posn`, `negn`, `z`. Thus, $x = \{x_{pos1}, x_{neg1}, \ldots, x_{posn}, x_{negn}, x_z\}$ and $y = \{y_{pos1}, y_{neg1}, \ldots, y_{posn}, y_{negn}, y_z\}$. Domains are defined by $D(x_{posi}) = \{0,1\}, \forall i$, $D(x_{negi}) = \{0,1\}, \forall i$, $D(x_z) = \{0\}$, $D(y_{posi}) = \{0\}, \forall i$, $D(y_{negi}) = \{0\}, \forall i$ and $D(y_z) = \{1\}$. We denote by `All` the set of all items. The confidence ratio c is set to 0.5.

The transactions table $\mathcal{D}$ is:

(i) `All` $\setminus$ {`z`} (n times)

(ii) $\texttt{All} \setminus \{\texttt{posi}\}, \forall v_i \in V$

(iii) $\texttt{All} \setminus \{\texttt{negi}\}, \forall v_i \in V$

(iv) $\texttt{All} \setminus \{\texttt{posi}, \texttt{negi}, \texttt{z}\}, \forall v_i \in V$ (2 times)

(v) $\texttt{All} \setminus \{it_1, it_2, it_3, \texttt{z}\}$, for each clause cl in F, where $it_i = \texttt{posj}$ if the ith literal in cl is v_j, $it_i = \texttt{negj}$ if the ith literal in cl is $\neg v_j$.

Suppose a formula F is satisfiable. Let us denote by S a solution of F. We construct the valid assignment on x, y such that $x_{posi} = 1$ and $x_{negi} = 0$ for each i such that $S[v_i] = 1$, and $x_{posi} = 0$ and $x_{negi} = 1$ for each i such that $S[v_i] = 0$. Bear in mind that x_z and y are already assigned as they have a singleton domain. By construction of $\mathcal{D}$, $x^{-1}(1) \cup y^{-1}(1)$ appears in n transactions (ii) and (iii). By construction again, $x^{-1}(1)$ appears in the n transactions where $x^{-1}(1) \cup y^{-1}(1)$ appears plus the n transactions (i). $x^{-1}(1)$ does not appear in any transaction (iv) because they all miss posi and negi for some i, whereas $x^{-1}(1)$ contains posi or negi for all i. Finally, as S satisfies F, $x^{-1}(1)$ does not appear in any transaction (v) because these transactions all miss at least the item of $x^{-1}(1)$ corresponding to the literal satisfying the clause. As a result, the rule $x^{-1}(1) \rightarrow y^{-1}(1)$ has confidence $\frac{n}{2n} = 0.5$, and our assignment on (x, y) satisfies the constraint $\textsc{Confident}_{\mathcal{D},c}(x, y)$.

Suppose now that A is a valid assignment on x, y satisfying the constraint $\textsc{Confident}_{\mathcal{D},c}(x, y)$. Remember that $y^{-1}(1)$ necessarily contains z, so $x^{-1}(1)$ does not. Hence, $x^{-1}(1)$ appears at least in the n transactions (i) where $y^{-1}(1)$ does not appear. Now, $y^{-1}(1)$ only appears in transactions (ii) and (iii) because it contains z. Thus, $x^{-1}(1)$ must appear in at least n transactions (ii) and (iii) to reach the confidence of 50%. For a given i, $x^{-1}(1)$ must contain at least one among posi and negi, otherwise the two corresponding transactions (iv) would cover $x^{-1}(1)$ and not $y^{-1}(1)$, making confidence impossible to reach. Thus, $x^{-1}(1)$ can (and must) appear in exactly n transactions (ii) and (iii), which means that for each i, exactly one among posi and negi is in $x^{-1}(1)$. We then can build the mapping from the assignment A on x, y to the instantiation S on $v_1, \ldots, v_n$ such that $S[v_i] = 1$ if $A[x_{posi}] = 1$, and $S[v_i] = 0$ if $A[x_{negi}] = 1$. We have n transactions (i-iv) covering $x^{-1}(1) \cup y^{-1}(1)$ and $2n$ covering $x^{-1}(1)$. As transactions (v) do not contain z, they must not cover $x^{-1}(1)$, otherwise confidence cannot be reached. As a result, for every transaction (v), $x^{-1}(1)$ necessarily contains at least one item (other than z) which is not in the transaction. By construction of transactions (v) and thanks to the mapping from A to S, this item corresponds to the truth value of a Boolean variable that satisfies the clause of F associated with the transaction. Therefore, F is satisfiable.

Consequently, deciding if there exists a valid assignment satisfying the constraint $\textsc{Confident}_{\mathcal{D},c}$ is NP-complete, and domain consistency on $\textsc{Confident}_{\mathcal{D},c}$ is NP-hard. □

Theorem 1 tells us that we cannot efficiently enforce domain consistency on $\textsc{Confident}_{\mathcal{D},c}$ unless $P = NP$. We thus propose a weaker propagation operated by a decomposition of $\textsc{Confident}_{\mathcal{D},c}$ using the global constraint $\textsc{CoverSize}$ [13], which is able to capture the frequencies of x and $z = x \cup y$. $\textsc{CoverSize}_{\mathcal{D}}(x, p)$ holds if and only if $p = |cover(x^{-1}(1))|$, where p is an additional variable internal to the model. The decomposition of $\textsc{Confident}_{\mathcal{D},c}$ is as follows.

$$\textsc{Confident}_{\mathcal{D},c}(x, y) \equiv \begin{cases} \textsc{CoverSize}_{\mathcal{D}}(x, p) \\ \textsc{CoverSize}_{\mathcal{D}}(x \cup y, q) \\ \frac{q}{p} \geq c \end{cases}$$

5 A CP model for computing MNRs

In this section we show how our CP-Rule model can be extended to only return MNRs.

Definition 2. (Minimal Non-redundant Rule (MNR) [2]) *Given a minimum support s and a minimum confidence c, a* minimal non-redundant association rule *$r : X \rightarrow Y$ is a valid rule such that there does not exist any rule $r' : X' \rightarrow Y'$ with $X' \subseteq X$, $Y \subseteq Y'$, $freq(r) = freq(r')$, $conf(r) = conf(r')$, and $r \neq r'$.*

Example 3. *From the transaction dataset presented in Table 1 and with $s = 20\%$ and $c = 20\%$. The rule $CD \rightarrow E$ is redundant to the rule $D \rightarrow CE$ as $D \subset CD$, $E \subset CE$, $freq(CD \rightarrow E) = freq(D \rightarrow CE) = 20\%$ and $conf(CD \rightarrow E) = conf(D \rightarrow CE) = 33.3\%$.*

There exists an interesting operational characterization of MNRs [2].

Proposition 1. ([2]) *An association rule $X \rightarrow Y$ is an MNR if and only if:*

$$\begin{cases} freq(X \rightarrow Y) \geq s \wedge conf(X \rightarrow Y) \geq c & (1) \\ X \text{ is a generator} & (2) \\ X \cup Y \text{ is closed} & (3) \end{cases}$$

According to Proposition 1, our CP-Rule model can be extended to a model able to extract MNRs by adding two constraints:

$$\textsc{mnRule}_{\mathcal{D},s,c}(x, y, z) = \begin{cases} \textsc{CP-Rule}_{\mathcal{D},s,c}(x, y, z) & (1) \\ \textsc{Generator}_{\mathcal{D}}(x) & (2) \\ \textsc{Closed}_{\mathcal{D}}(z) & (3) \end{cases}$$

In the model mnRule,

(1) CP-Rule ensures that the rule $x^{-1}(1) \to y^{-1}(1)$ is valid;

(2) is a global constraint that ensures that the itemset $x^{-1}(1)$ is a generator;

(3) is a global constraint that ensures that the itemset $z^{-1}(1)$ is closed.

The new global constraint Generator is introduced in the rest of this section.

Definition 3. (Generator constraint) *Let x be a vector of Boolean variables and $\mathcal{D}$ be a dataset. The global constraint* Generator$_{\mathcal{D}}(x)$ *holds if and only if $x^{-1}(1)$ is a generator.*

The propagator we propose for the Generator constraint is based on the following property of generators.

Proposition 2. *Given two itemsets P and Q, if P is not a generator and $P \subsetneq Q$, then Q is not a generator.*

Proof. Derived from Theorem 2 in [3], where we set frequency to 0. □

Algorithm. The propagator Filter-Generator for the global constraint Generator is presented in Algorithm 1. Filter-Generator takes as input the variables x. Filter-Generator starts by computing the cover of the itemset $x^{-1}(1)$ and stores it in `cover` (line 3). Then, for each item $j \in x^{-1}(1)$, Filter-Generator computes the cover of the subset $x^{-1}(1) \setminus \{j\}$, and stores it in `cov[j]` (lines 4-5). Filter-Generator can then remove items i that cannot belong to a generator containing $x^{-1}(1)$. To do that, for every item j in $x^{-1}(1) \cup \{i\}$, we compare the cover of $x^{-1}(1) \cup \{i\}$ (i.e., $\texttt{cover} \cap cover(i)$) to the cover of $x^{-1}(1) \cup \{i\} \setminus \{j\}$ (i.e., $\texttt{cov[j]} \cap cover(i)$) (line 8). If they have equal size (i.e., same frequency), we remove i from the possible items, that is, we remove 1 from $dom(x_i)$ and break the loop (line 9).

Theorem 2. *The propagator* Filter-Generator$_{\mathcal{D}}$ *enforces domain consistency.*

Proof. We first prove that the value 0 for a variable x_i such that $i \notin (x^{-1}(1) \cup x^{-1}(0))$ always belongs to a solution of the constraint Generator, and so cannot be pruned by domain consistency. Suppose $i \notin x^{-1}(1) \cup x^{-1}(0)$. If $x^{-1}(1)$ is a generator, removing the value 0 from $dom(x_i)$ increases $x^{-1}(1)$ to $x^{-1}(1) \cup \{i\}$, and then $x^{-1}(1)$ cannot be returned as a generator, which contradicts the hypothesis. Suppose now that

Algorithm 1: Filter-Generator$_{\mathcal{D}}$ (x)

```
1 InOut: $x = \{x_1 \dots x_n\}$: Boolean item variables;
2 begin
3     cover ← $cover(x^{-1}(1))$;
4     foreach $j \in x^{-1}(1)$ do
5         cov[j] ← $cover(x^{-1}(1) \setminus \{j\})$;
6     foreach $i \notin x^{-1}(1) \cup x^{-1}(0)$ do
7         foreach $j \in x^{-1}(1) \cup \{i\}$ do
8             if $|\texttt{cover} \cap cover(i)| = |\texttt{cov[j]} \cap cover(i)|$ then
9                 $dom(x_i) \leftarrow dom(x_i) \setminus \{1\}$; break;
```

$x^{-1}(1)$ is not a generator. We know from Proposition 2 that for any $Q \supsetneq x^{-1}(1)$, Q is not a generator. Thus, $x^{-1}(1) \cup \{i\}$ cannot belong to any generator, and value 0 cannot be pruned from $dom(x_i)$.

We now prove that Filter-Generator prunes value 1 from $dom(x_i)$ exactly when i cannot belong to a generator containing $x^{-1}(1)$. Suppose value 1 of x_i is pruned by Filter-Generator. This means that the test in line 8 was true, that is, there exists a sub-itemset of $x^{-1}(1) \cup \{i\}$ with the same frequency as $x^{-1}(1) \cup \{i\}$. Thus, by definition, $x^{-1}(1) \cup \{i\}$ does not belong to any generator. Suppose now that value 1 of x_i is not pruned. From line 8, we deduce that there does not exist any subset of $x^{-1}(1) \cup \{i\}$ with the same frequency as $x^{-1}(1) \cup \{i\}$. Thus $x^{-1}(1) \cup \{i\}$ is a generator and value 1 of x_i is domain consistent. □

Theorem 3. *Given a transaction dataset $\mathcal{D}$ of n items and m transactions, the algorithm* Filter-Generator$_{\mathcal{D}}$ *has an $O(n^2 \times m)$ time complexity.*

Proof. Computing the size of the cover of an itemset is in $O(n \times m)$. Line 5 is called at most n times, leading to a time complexity of $O(n^2 \times m)$. The test at line 8 is done at most n^2 times. The covers of $x^{-1}(1) \cup \{i\}$ and $x^{-1}(1) \cup \{i\} \setminus \{j\}$ at line 8 are computed in $O(m)$ thanks to the `cover` and `cov` data structures. Thus, the time complexity of lines 6-9 is bounded above by $O(n^2 \times m)$. As a result, Filter-Generator has an $O(n^2 \times m)$ time complexity. □

Note that without the use of the `cov` structure (that is, by recomputing $cover(x^{-1}(1) \setminus \{j\})$ at each execution of the loop at line 6), the time complexity becomes $O(n^3 \times m)$. However, this version is less memory consuming and can be more efficient in practice. It is also important to stress that domain consistency on Generator does not depend on $x^{-1}(0)$. Thus, Filter-Generator is not called during the solving process when a variable is instantiated to zero.

Table 2: Dataset Characteristics.

Dataset	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{I}\rvert$	$\overline{\lvert\mathcal{T}\rvert}$	$\rho(\%)$	Type of Data
Zoo	101	36	16	44	Animals data
Vote	435	48	16	33	Vote data
Anneal	812	93	42	45	Anneal data
Chess	3,196	75	37	49	Game steps
Mushroom	8,124	119	23	19	Species of mushrooms
Connect	67,557	129	43	33	Game steps
T10	100,000	1,000	10	1	Synthetic dataset
T40	100,000	1,000	40	4	Synthetic dataset
Pumsb	49,046	7,117	74	1	Census data
Retail	88,162	16,470	10	0.06	Retail market basket data

T10 = T10I4D100K   T40 = T40I10D100K

6 Experimental Evaluation

We made several experiments to evaluate our CP model for mining pure and constrained association rules. We compared it to the state of the art approaches.

6.1 Benchmark datasets. We selected several real-sized datasets from the FIMI repository.[1] These datasets have various characteristics representing different application domains. Table 2 reports for each dataset the number of transactions $|\mathcal{T}|$, the number of items $|\mathcal{I}|$, the average size of transactions $\overline{|\mathcal{T}|}$, the density ρ (i.e., $\overline{|\mathcal{T}|}/|\mathcal{I}|$), and its application domain. The datasets are presented by increasing size $|\mathcal{I}| \cdot |\mathcal{T}|$. We selected datasets of various size and density. Some datasets, such as Zoo and Chess, are very dense (resp. 44% and 49%). Others, such as T10 and Retail, are very sparse (resp. 1% and 0.06%). The sizes of these datasets vary from around $4,000$ to more than 10^9.

6.2 Experimental protocol. The implementation of our CP models and constraint propagators were carried out in the Oscar solver using Scala.[2] The code is publicly available at https://gite.lirmm.fr/belaid/cp4ar. For propagating the Confident constraint, we have used the decomposition of Confident presented in Section 3. In the mnRule model presented in Section 5, we have used the global constraint Generator presented in the same section, and to ensure closedness, we have used the constraint CoverClosure introduced in [13]. After a few preliminary tests, we decided to use $\langle x, y, z\rangle$ as variable ordering heuristics and *smallest value first* as value ordering heuristics.

We compared our CP approach to the Eclat-Z specialized algorithm for extracting ARs and MNRs [15], and to a SAT-based approach [5, 6]. We used the implementation of Eclat-Z publicly available in the coron Data Mining Platform.[3] In all the instances presented in the following experiments, the minimum confidence has been fixed to 90% so that we focus on highly confident rules. All experiments were conducted on an Intel core i7, 2.2Ghz with a RAM of 8Gb and a timeout of one hour.

6.3 Mining association rules. Our first experiment compares Eclat-Z to the SAT and CP approaches (denoted by sat and cp in the following) for extracting the whole set of ARs. For each dataset, an instance is characterized by its minimum support. For instance, Zoo_50 denotes the instance of the Zoo dataset with a minimum support of 50% (and always a minimum confidence of 90%). For each dataset, we have selected three instances according to the number of solutions. The first instance has less than $1,000$ ARs, the second instance has between $100,000$ and one million ARs, and the third instance has more than one million ARs. The only exception is the very large and sparse dataset Retail, for which we could not reach large numbers of solutions. Table 3 reports the CPU time, in seconds, for each approach on each selected instance. We also report the total number of ARs (#Tot) for each instance.[4]

The first observation that we can draw from Table 3 is that, as expected, the specialized algorithm Eclat-Z performs very well. However, the CP approach is very competitive too, and sometimes faster than Eclat-Z (see the first instance of each dataset, where only a few valid ARs exist). The explanation for this good behavior of cp is the strength of constraint propagation to rule out inconsistent parts of the search space. On the contrary, on an instance as loose as Pumsb_80, with $49,000$ transactions, $7,000$ items, and around 20 million solutions, the CP solver is almost reduced to an enumerating process.

Concerning the SAT approach, on small and middle-sized datasets (i.e., from Zoo to Connect), we observe that sat performs reasonably well, although being slower than cp on almost all the instances. On larger and larger datasets, the results of sat become worse and worse. On very large datasets, sat reaches the one hour timeout for T10 and T40 instances and out-of-memory on all the instances of Pumsb and Retail. The out-of-memory state is due to the fact that sat generates huge propositional formulas to represent the dataset and the constraints. For a dataset of n items and m transactions, sat generates a formula of $(2n+2m)$ variables and a number of clauses bounded below by $(n+m+nm)$. On the instances where we observe an out-of-memory, sat generates formulas of size ranging from 14Gb to 63Gb.

Finally, it is important to note that even when cp

[1]http://fimi.ua.ac.be/data/

[2]bitbucket.org/oscarlib/oscar/

[3]http://coron.loria.fr

[4]This number is computed by releasing the timeout.

Table 3: ECLAT-Z vs SAT vs CP for extracting ARs (time in seconds)

Instances	ECLAT-Z	SAT	CP	#TOT
`Zoo_50`	0.07	**0.02**	0.04	292
`Zoo_30`	**0.40**	0.92	0.82	198,971
`Zoo_5`	**24.99**	106.52	43.45	30,792,317
`Vote_35`	**0.04**	0.32	0.05	271
`Vote_10`	**1.81**	255.52	1.79	419,204
`Vote_5`	**5.66**	1054.81	6.43	2,075,212
`Anneal_96`	0.08	0.19	**0.06**	798
`Anneal_90`	**0.28**	1.11	1.17	174,710
`Anneal_80`	**93.96**	467.83	208.87	84,589,753
`Chess_95`	0.08	0.56	**0.07**	474
`Chess_80`	**0.62**	5.97	2.16	349,298
`Chess_60`	**15.98**	260.21	68.74	17,522,446
`Mushroom_50`	**0.10**	3.78	**0.10**	469
`Mushroom_21.5`	**0.59**	51.87	1.43	112,826
`Mushroom_10`	**27.81**	686.39	50.67	14,331,056
`Connect_99`	0.64	26.90	**0.04**	70
`Connect_94`	**0.99**	95.97	7.70	201,928
`Connect_90`	**3.03**	1126.91	125.48	3,640,704
`T10_0.5`	**0.29**	TO	2.71	532
`T10_0.1`	**1.63**	TO	9.92	160,171
`T10_0.02`	**15.12**	TO	63.35	1,303,932
`T40_1.6`	**1.35**	TO	5.36	648
`T40_1.3`	**1.73**	TO	13.16	101,588
`T40_0.1`	TO	TO	TO	$> 7.10^9$
`Pumsb_96`	1.31	OOM	**0.11**	338
`Pumsb_89`	**2.07**	OOM	8.39	135,677
`Pumsb_80`	**34.35**	OOM	1567.72	19,749,382
`Retail_0.5`	1.59	OOM	**0.52**	37
`Retail_0.3`	**1.74**	OOM	2.27	75
`Retail_0.1`	**2.53**	OOM	58.76	255

TO: **timeout** OOM: **out-of-memory**

Table 4: ECLAT-Z vs SAT vs CP for extracting MNRs (time in seconds)

Instances	ECLAT-Z	SAT	CP	#TOT
`Zoo_50`	0.04	**0.03**	0.07	177
`Zoo_30`	0.39	**0.08**	0.16	2,262
`Zoo_5`	8.21	**0.47**	0.61	13,988
`Vote_35`	**0.05**	0.38	0.08	271
`Vote_10`	10.95	29.08	**2.02**	259,445
`Vote_5`	47.40	49.14	**3.23**	505,225
`Anneal_96`	**0.05**	0.39	**0.05**	87
`Anneal_90`	**0.18**	0.60	0.38	4,825
`Anneal_80`	5.70	1.48	**1.44**	46,871
`Chess_95`	0.09	1.90	**0.08**	465
`Chess_80`	**0.92**	7.64	1.78	191,158
`Chess_60`	70.94	254.87	**25.45**	4,633,266
`Mushroom_50`	0.09	30.57	**0.06**	105
`Mushroom_21.5`	0.42	72.88	**0.26**	2,211
`Mushroom_10`	3.45	205.46	**0.78**	11,421
`Connect_99`	0.62	828.00	**0.06**	70
`Connect_94`	**1.10**	863.17	3.94	51,754
`Connect_90`	**7.28**	1107.33	19.39	322,838
`T10_0.5`	**0.53**	OOM	6.40	532
`T10_0.1`	42.58	OOM	**21.01**	147,549
`T10_0.02`	1131.39	OOM	**195.95**	257,318
`T40_1.6`	**3.54**	OOM	13.93	648
`T40_1.3`	**8.97**	OOM	22.79	101,588
`T40_0.1`	TO	OOM	TO	$> 10^9$
`Pumsb_96`	1.44	OOM	**0.21**	277
`Pumsb_89`	**1.99**	OOM	7.43	69,222
`Pumsb_80`	**71.96**	OOM	355.40	3,668,125
`Retail_0.5`	**2.07**	OOM	2.47	37
`Retail_0.3`	**2.01**	OOM	7.88	75
`Retail_0.1`	**12.67**	OOM	120.94	253

TO: **timeout** OOM: **out-of-memory**

reaches the timeout, it returns solutions on the fly before the timeout. This is because CP, as opposed to ECLAT-Z, does not need to build any complex data structure before starting the search for ARs. On `T40_0.1`, CP returns the first solution in only 1.25 seconds and returns more than 390 million solutions before the timeout, whereas ECLAT-Z does not return any AR to the user before having computed its whole structure, that is, no solution before the timeout on `T40_0.1`.

6.4 Mining MNRs. Our second experiment compares ECLAT-Z to SAT and CP for extracting MNRs. Table 4 reports the CPU time, in seconds, for each approach on each selected instance. We also report the total number of MNRs (#TOT).

SAT wins only on the smallest dataset (i.e., `Zoo` instances). For larger datasets, the encoding of the MNR constraints in SAT can take three times more space than the encoding of the original AR problem and then can lead to an out-of-memory state. For instance, SAT encodes the AR problem on a `Retail` instance with a formula of 63Gb. When moving to the MNR problem, we reach 187Gb.

ECLAT-Z can take significantly more time to enumerate MNRs than ARs. On the `T10_0.02` instance, it is easier for ECLAT-Z to enumerate more than 1 million ARs (15.12 seconds) than to enumerate 257 thousands MNRs (1131.39 seconds). The opposite is observed on CP. Such a performance is mainly due to the constraints GENERATOR and COVERCLOSURE that provide the CP solving process with more propagation to remove more inconsistent values in the search space. This is especially true when the number of non-solutions to prune is important. Nevertheless, the `Retail` instances contradict this trend. This is explained by the fact that on `Retail` almost all ARs are MNRs whereas in average on all our instances only 7% of the ARs are MNRs. Hence, on `Retail` the constraints GENERATOR and COVERCLOSURE are redundant to the model for ARs (CP-RULE). Thus, the propagators of GENERATOR and COVERCLOSURE do not participate to the reduction of the search space. They just waste time. For instance, CP explores exactly the same search space ($2,750$ nodes) on `Retail_0.3` to extract ARs and MNRs (because all ARs are MNRs).

Again, on the timeout instance `T40_0.1`, CP returns its first solution in 1.23 seconds and returns more than 160 million MNRs before the timeout. ECLAT-Z is not able to return any MNR before the timeout.

6.5 Mining constrained association rules. In many practical cases, the user asks for patterns satisfying some additional constraints. Such user's constraints are handled by data mining researchers, whenever possible, with (i) a pre-processing step reducing the dataset; (ii) a filtering integrated in the specialized algorithm (which requires to implement an ad hoc algorithm for each user's constraint); (iii) a post-processing step to filter out the undesirable patterns.

In this section, we present experiments that show the expressiveness of our CP approach in taking into account user's constraints, and the power of CP to solve such combinatorial problems. We illustrate with mandatory/forbidden-item constraints and with cardinality constraints on the body or the head of the rule. We compare our CP approach with SAT and Eclat-Z-pp (Eclat-Z with a post-processing step filtering out the rules not satisfying the user's constraints). We selected the instances having more than one million ARs in Table 3 (i.e., $\#Tot \geq 10^6$).

Mining rules with constraints on items. The user can ask for rules with some specific items in the body and/or the head. In the same way, a user can ask for rules not involving a particular set of items. For instance, we can want to extract rules between electronics and cleaning items only. Or to extract rules outside food items. We perform an experiment on the following user query:

Q_1 : *Given two sets of items $\mathcal{F}$ and $\mathcal{M}$, extract ARs not containing $\mathcal{F}$ in the body, and containing $\mathcal{M}$ in the head.*

Q_1 can easily be expressed in CP with the following constraints:

$$\text{CP-Rule}_{\mathcal{D},s,c}(x, y, z) \wedge Forb_{\mathcal{F}}(x) \wedge Mand_{\mathcal{M}}(y)$$

where, $Forb_{\mathcal{F}}(x) \equiv (\forall i \in \mathcal{F} : x_i = 0)$ and $Mand_{\mathcal{M}}(y) \equiv (\forall i \in \mathcal{M} : y_i = 1)$.

For our experiment, we generate sets $\mathcal{F}$ and $\mathcal{M}$ of the same size. We randomly select a pair of items and we put one in $\mathcal{F}$ and one in $\mathcal{M}$. We repeat the process until the number of solutions to Q_1 falls under a very low threshold. Table 5 reports the penultimate step, that is, the $\mathcal{F}$ and $\mathcal{M}$ where the number of solutions is the smallest above 10. We made an exception with T10_0.02, that we will use to illustrate what happens on an instance with no solution. Table 5 reports the CPU time of Eclat-Z-pp, SAT and CP on Q_1. We also report the number of solutions to Q_1 (#Tot).

The main observation is that the declarative approaches SAT and CP outperform Eclat-Z-pp. Eclat-Z-pp extracts all ARs and filters out the rules the user is not asking for. For Anneal_80, Eclat-Z-pp spends 14 minutes to extract and save more than 84 million ARs. The post-processing step spends 10 minutes to filter out this huge number of ARs and to return the only 21 solutions of Q_1. With SAT and CP, the 21 solutions are returned in less than one second.

Table 5: Eclat-Z-pp vs SAT vs CP on Q_1 (time in seconds).

Instances	$\|\mathcal{F}\|$	$\|\mathcal{M}\|$	Eclat-Z-pp	SAT	CP	#Tot
Zoo_5	11	11	535.69	**0.02**	**0.02**	15
Vote_5	7	7	38.57	0.16	**0.04**	14
Anneal_80	12	12	1454.23	0.19	**0.01**	21
Chess_60	9	9	292.59	0.54	**0.07**	32
Mushroom_10	10	10	263.93	19.82	**0.02**	21
Connect_90	9	9	61.04	24.22	**0.02**	19
T10_0.02	10	10	84.04	TO	**0.02**	0
T40_0.1	10	10	TO	TO	**0.06**	19
Pumsb_80	12	12	765.67	OOM	**0.02**	23

TO: **timeout** OOM: **out-of-memory**

For the SAT approach, the forbidden and mandatory constraints can easily be encoded with monomials (i.e., unit clauses) that reduce the search space and speed-up the resolution. However, when the size of the dataset increases, the performance of SAT decreases. SAT cannot solve T40_0.1 within the time out. In addition, SAT faces again the problem of the size of the formulas: Pumsb_80 has an out-of-memory.

On the instance with no solution T10_0.02, SAT cannot prove that no solution exists within the timeout. Eclat-Z-pp spends more than 1 minute to extract more than 1 million useless solutions and filter them out. CP proves the absence of solutions in 0.02s.

Mining rules with cardinality constraints. The user can also ask for rules with particular sizes of body or head. We performed an experiment with the following query:

Q_2 : *Extract ARs with a body not exceeding a size of ub and a head of a minimum size of lb.*

Q_2 can easily be expressed in CP with:

$$\text{CP-Rule}_{\mathcal{D},s,c}(x, y, z) \wedge atMost_{ub}(x) \wedge atLeast_{lb}(y)$$

where $atMost_{ub}(x) \equiv \sum_{i \in \mathcal{I}} x_i \leq ub$ and $atLeast_{lb}(y) \equiv \sum_{i \in \mathcal{I}} y_i \geq lb$.

For each instance, we select a lower-bound lb and an upper-bound ub in order to have at least 10 solutions to Q_2. For that, we use two scenarios. The first scenario starts by selecting the largest value for lb for which there exist at least 10 ARs with a head of minimum size lb. Then, we take for ub the size of the smallest body for which there still exist at least 10 solutions. The second scenario selects the smallest value for ub for which there exist at least 10 ARs with a body not exceeding this value. Then, we take for lb the size of the largest head for which there still exist at least 10 solutions. Again, for T10_0.02, we select lb and ub in order to illustrate the case of no solution. Table 6 compares Eclat-Z-pp,

Table 6: ECLAT-Z-PP vs SAT vs CP on Q_2 (time in seconds).

Instances	*ub*	*lb*	ECLAT-Z-PP	SAT	CP	#TOT
Zoo_5	2	11	479.26	3.92	**0.36**	27
Zoo_5	1	9	491.48	0.17	**0.06**	12
Vote_5	4	8	37.69	282.25	**0.66**	13
Vote_5	1	2	38.49	1.41	**0.05**	23
Anneal_80	2	13	1567.48	1.14	**0.26**	76
Anneal_80	1	12	1622.19	0.53	**0.15**	73
Chess_60	2	9	280.60	2.17	**0.20**	20
Chess_60	1	8	284.22	1.07	**0.08**	24
Mushroom_10	1	11	249.00	47.52	**0.07**	14
Connect_90	1	11	61.80	30.41	**0.26**	12
T10_0.02	1	11	84.47	TO	**5.44**	0
T40_0.1	1	11	TO	TO	**8.33**	39
Pumsb_80	1	12	741.49	OOM	**0.34**	32

TO: **timeout** OOM: **out-of-memory**

SAT and CP acting on the query Q_2. We report the CPU time, in seconds. For each instance, we also report the total number of solutions (#TOT).

We observe that CP wins on all instances. SAT reaches the one hour timeout on two instances whereas these two instances are solved by CP in less than 10 seconds. Once again, SAT faces an out-of-memory state on `Pumsb_80`.

When comparing CP to ECLAT-Z-PP we can observe that for the instances having more than 10 million ARs we have a significant gap in terms of CPU time. CP on `Anneal_80` returns the 73 solutions in less than one second, whereas ECLAT-Z-PP spends 13 minutes on the post-processing step.

For the instance with no solution `T10_0.02`, SAT cannot prove there is no solution before the time out. ECLAT-Z-PP needs a post-processing of more than 1 minute to deal with the 1 million extracted ARs. CP proves there is no solution in less than 6 seconds.

7 Conclusion

We have introduced a constraint programming model for mining association rules. This model requires the global constraint CONFIDENT for ensuring the confidence of a rule. We proved that enforcing domain consistency on this constraint is NP-hard, ruling out the possibility of a polynomial domain consistency propagator for CONFIDENT (unless $P = NP$). We thus proposed a decomposition of the constraint. Our CP model can easily be extended to extract different types of rules. For instance, minimal non-redundant rules can be extracted thanks to a new global constraint GENERATOR for generators, for which we have proposed a polynomial propagator achieving domain consistency. We also can capture any kind of user's constraint, such as mandatory or forbidden items, cardinality, etc. We have empirically evaluated our CP approach for extracting association rules with or without additional user's constraints. We have shown that the CP approach can solve huge datasets as opposed to the SAT approach. ECLAT-Z needs to extract the whole set of rules before applying a post-processing step and is thus not able to return any solution before the rules extraction process has finished.

References

[1] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *Acm sigmod record*, volume 22, pages 207–216. ACM, 1993.

[2] Y. Bastide, N. Pasquier, R. Taouil, G. Stumme, and L. Lakhal. Mining minimal non-redundant association rules using frequent closed itemsets. *Computational Logic*, pages 972–986, 2000.

[3] Y. Bastide, R. Taouil, N. Pasquier, G. Stumme, and L. Lakhal. Mining frequent patterns with counting inference. *SIGKDD Explorations*, 2(2):66–75, 2000.

[4] C. Bessiere, E. Hebrard, B. Hnich, and T. Walsh. The complexity of reasoning with global constraints. *Constraints*, 12(2):239–259, 2007.

[5] A. Boudane, S. Jabbour, L. Sais, and Y. Salhi. A sat-based approach for mining association rules. In *IJCAI*, pages 2472–2478. AAAI Press, 2016.

[6] A. Boudane, S. Jabbour, L. Sais, and Y. Salhi. Enumerating non-redundant association rules using satisfiability. In *PAKDD*, pages 824–836. Springer, 2017.

[7] L. De Raedt, T. Guns, and S. Nijssen. Constraint programming for itemset mining. In *KDD*, pages 204–212. ACM, 2008.

[8] M. Kryszkiewicz. Representative association rules. In *PAKDD*, pages 198–209. Springer, 1998.

[9] M. Kryszkiewicz. Representative association rules and minimum condition maximum consequence association rules. *PKDD*, pages 361–369, 1998.

[10] N. Lazaar, Y. Lebbah, S. Loudni, M. Maamar, V. Lemière, C. Bessiere, and P. Boizumault. A global constraint for closed frequent pattern mining. In *CP*, pages 333–349. Springer, 2016.

[11] D. Nguyen and B. Vo. Kse 2013. In *Knowledge and Systems Engineering*, pages 307–318. Springer, 2014.

[12] F. Rossi, P. Van Beek, and T. Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[13] P. Schaus, J. O. Aoga, and T. Guns. Coversize: A global constraint for frequency-based itemset mining. In *CP*, pages 529–546. Springer, 2017.

[14] L. Szathmary, A. Napoli, and S. O. Kuznetsov. Zart: A multifunctional itemset mining algorithm. In *CLA*, pages 26–37, 2007.

[15] L. Szathmary, P. Valtchev, A. Napoli, and R. Godin. An efficient hybrid algorithm for mining frequent closures and generators. In *CLA*, pages 47–58, 2008.

[16] M. Wojciechowski and M. Zakrzewicz. Dataset filtering techniques in constraint-based frequent pattern mining. *Pattern detection and discovery*, pages 301–318, 2002.